Отчёт по лабораторной работе №10

Борунов Семён

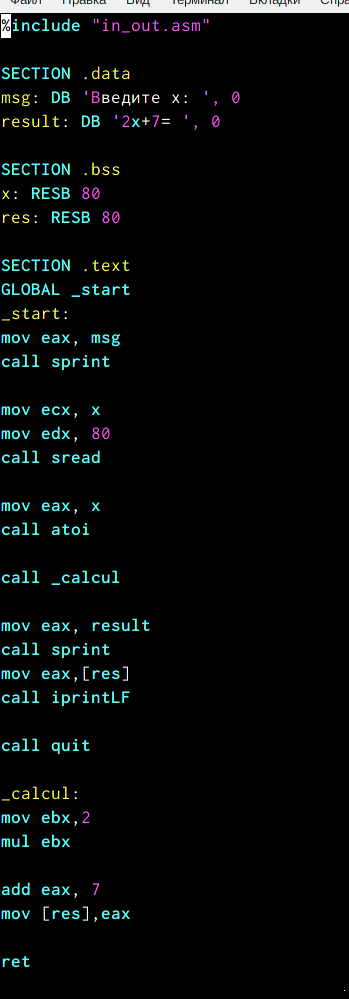
Содержание

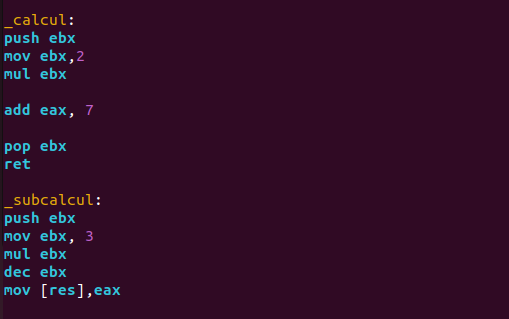
# 1 Цель работы

Освоить работу с подпрограммами и отладчиком gdb.

# 2 Выполнение лабораторной работы

Создадим рабочую директорию и файл. Запишем туда программу из листинга, исправив опечатки. (рис. **¿fig:001?**)



напишем программу, имитирующую сложную функцию. Функции назовем \_calul и *subcalcul*(рис. **¿fig:002?**) 

Проверим ее работу (рис. [-fig. 1)

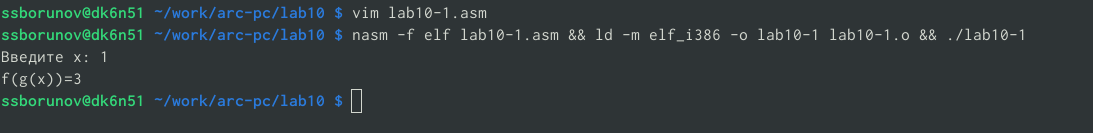
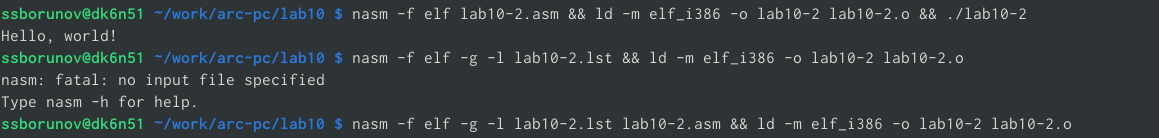
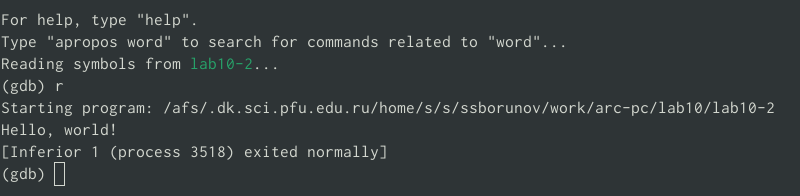


Рис. 1: Название рисунка

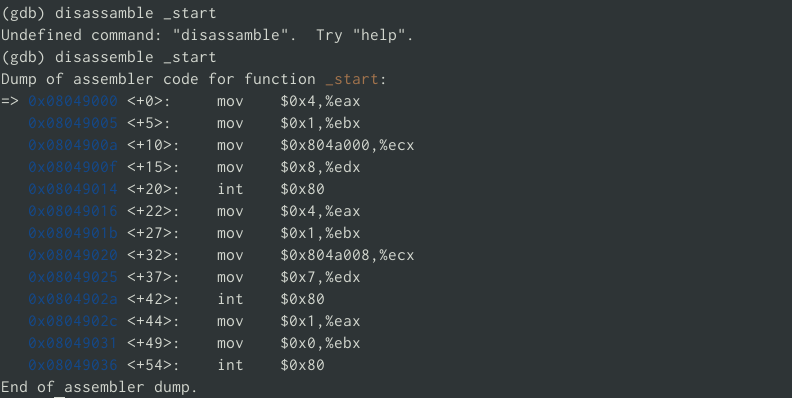
Создадим файл lab10-2.asm и посмотрим, как она работает. Так же проассемблируем его с другими ключами, чтобы была возможность открыть этот файл через gdb. (рис. **¿fig:004?**)



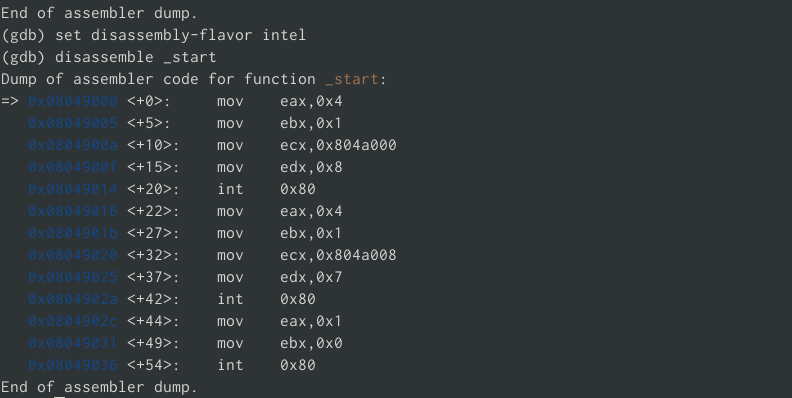
Откроем lab10-2 с помощью gdb. Запустим ее там(рис. **¿fig:005?**)



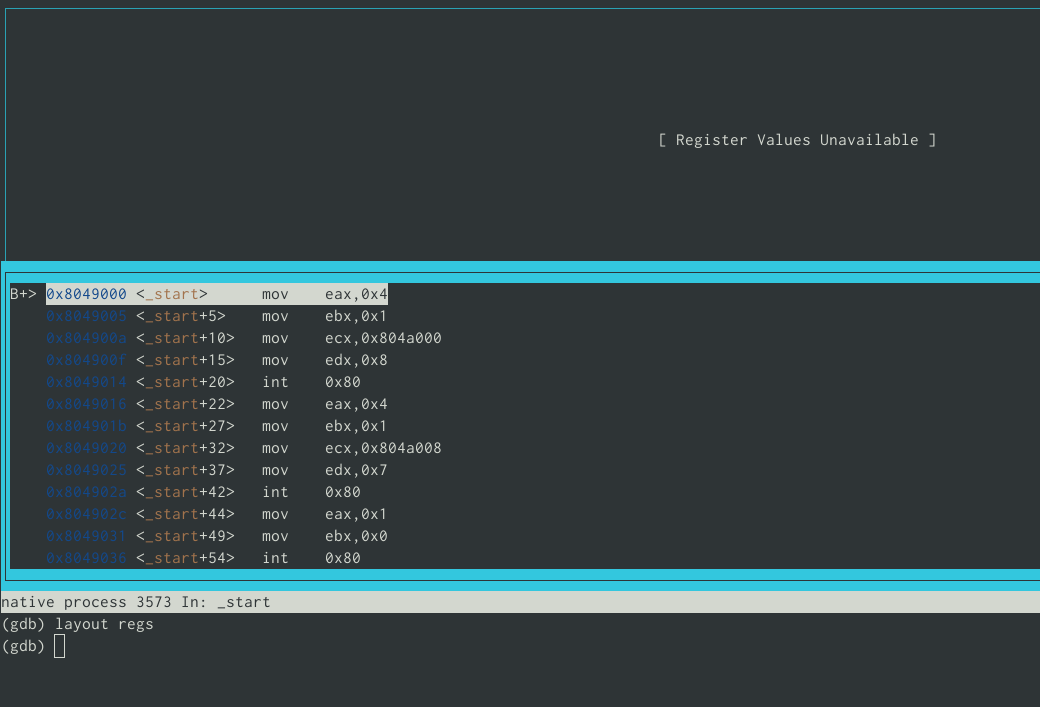
Поставим точку останова( breakpoint) на метке \_start. Посмотрим дизассемеблированный код, начиная с этой метки. (рис. **¿fig:006?**)



Так же посмотрим как выглядит дизассемблированный код c синтаксисом Intel (рис. **¿fig:007?**)

 В представлении ATT в виде 16-ричного числа записаны первые аргументы всех комманд, а в представлении intel так записываются адреса вторых аргумантов.

включим режим псевдографики, с помощью которго отбражается код программы и содержимое регистров(рис. **¿fig:008?**)



Посмотрим информацию о наших точках останова. Сделать это можно коротко командой i b (рис. 2)

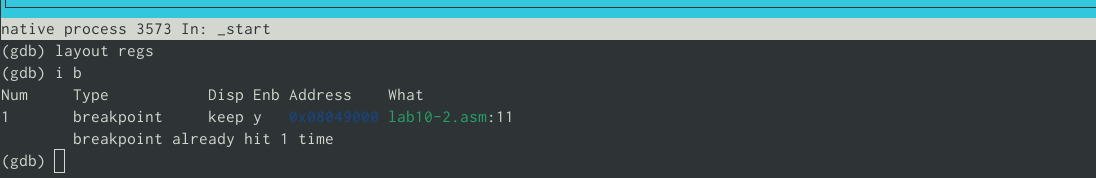
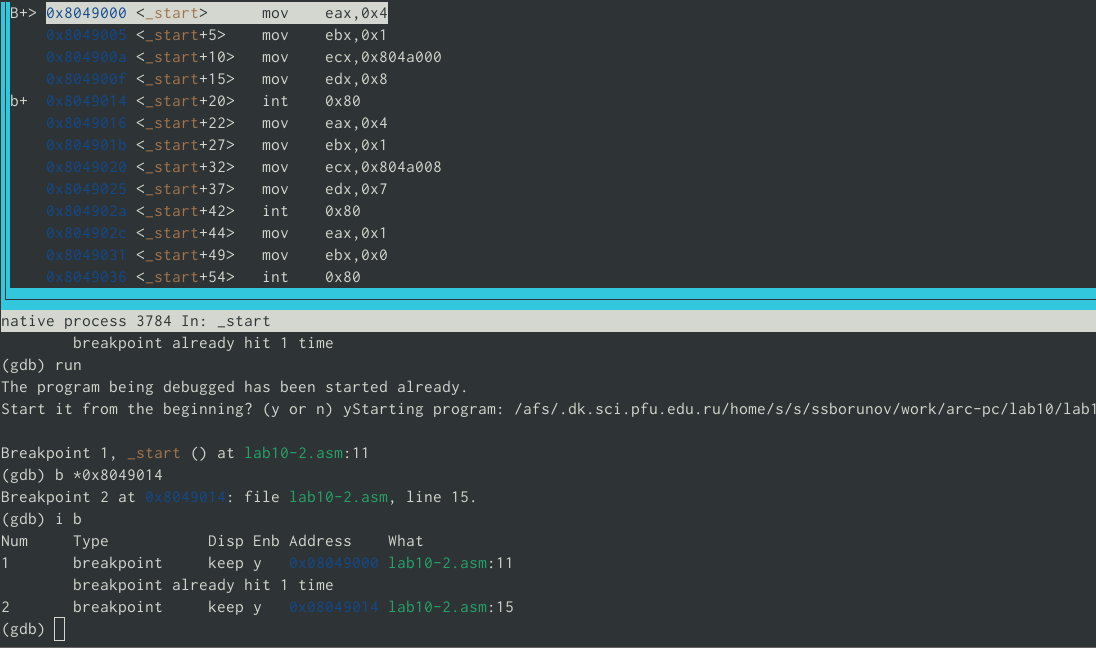
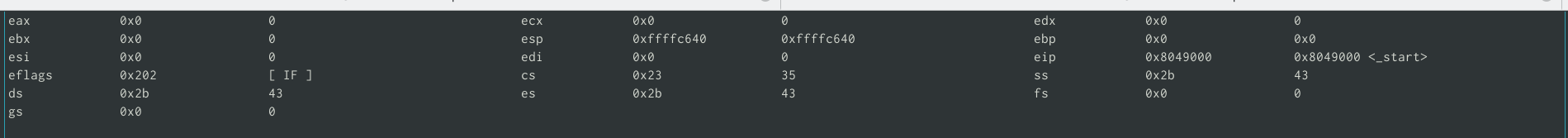


Рис. 2: Название рисунка

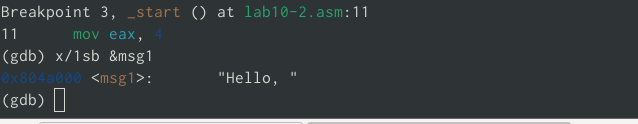
добавим еще одну точку останова, но сделаем это по адресу (рис. **¿fig:010?**)

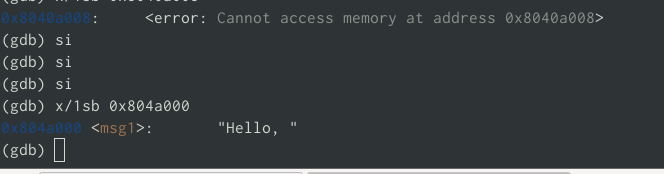


Так же можно выводить значения регистров. Делается это командой i r. Псевдографика предствалена на (рис. **¿fig:011?**)



В отладчике можно вывести текущее значение переменных. Сделать это можно например по имени (рис. **¿fig:012?**) или по адресу (рис. **¿fig:013?**)

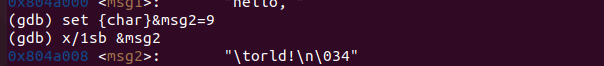




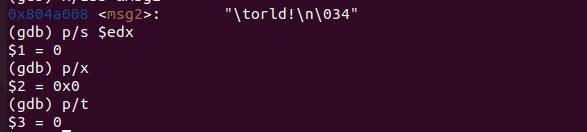
Так же отладчик позволяет менять значения переменных прямо во время выполнения программы (рис. **¿fig:014?**)

[](/home/ssborunov/work/study/2022-2023/Arch-pc/study\_2022-2023\_arh-pc/labs/lab10/report/image/Screenshot from 2022-12-16 13-51-33.png{ #fig:014 width=70% }

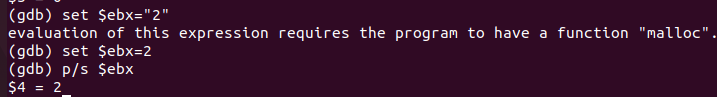
Здесь тоже можно обращаться по адресам переменных(рис. **¿fig:015?**). здесь был заменен первый символ переменной msg2 на символ отступа.



Выоводить можно так же содержимое регисторов. Выведем значение edx в разных форматах: строчном, 16-ричном, двоичном(рис. **¿fig:016?**)

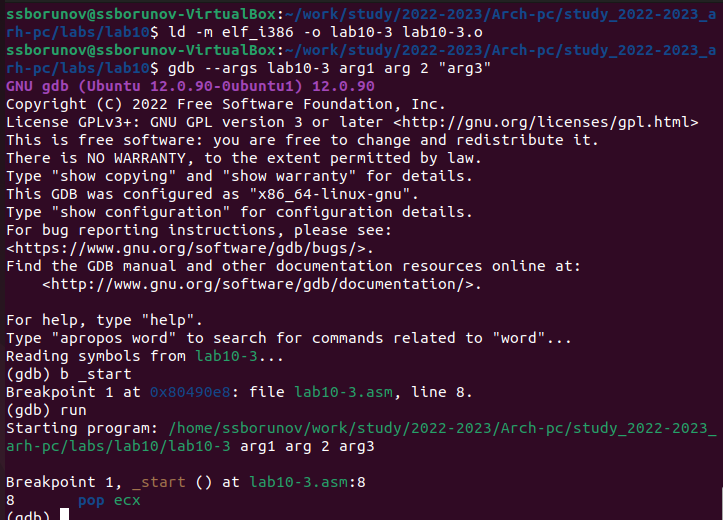


Как и переменным, регистрам можно задавать значения.(рис. **¿fig:017?**)

 Однако при попытке задать строчное значение, происходит ошибка.

Завершим работу в gdb командами continue, она закончит выполнение программы, и exit, она завершит сеанс gdb.

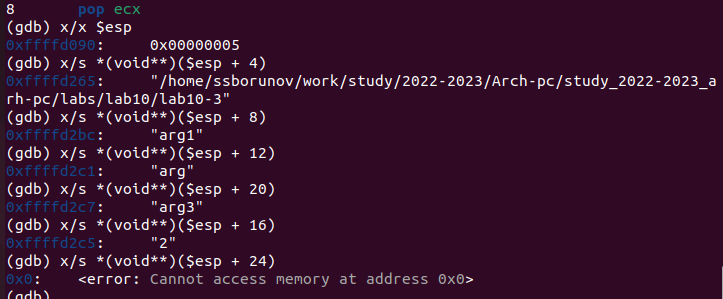
Скопируем файл из лабораторной 9, переименуем и создадим исполняемый файл. Откроем отладчик и зададим аргументы. Создадим точку останова на метке \_start и запустим программу(рис. **¿fig:018?**)



Посмотрим на содержимое того,что расположено по адрессу, находящемуся в регистре esp (рис. **¿fig:019?**)

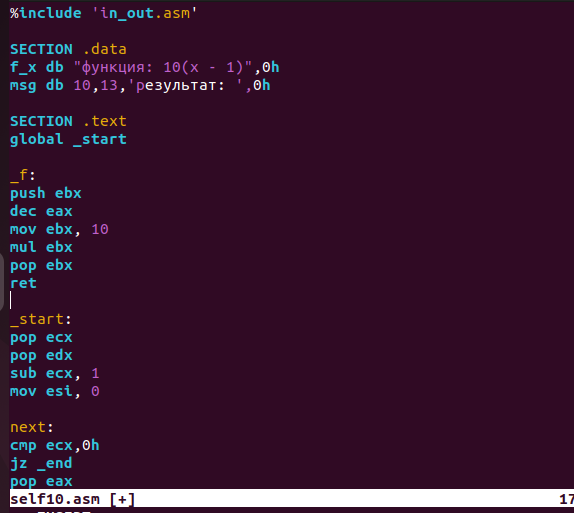


Далее посмотрим на все остальные аргументы в стеке. Их адреса распологаются в 4 байтах друг от друга(именно столько заниемает элемент стека) (рис. **¿fig:020?**)

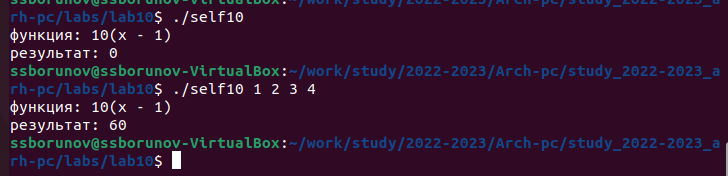


# 3 Задания для самостоятельной работы

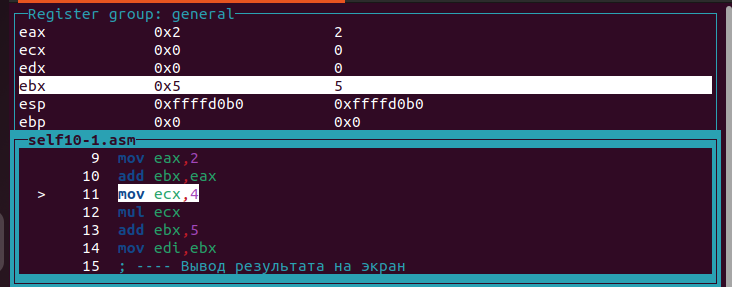
Программа из лабороторной 9, но с использованием подпрограмм (рис. **¿fig:021?**)

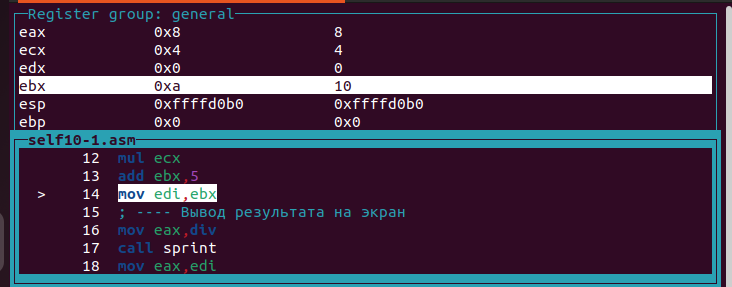


и проверка ее работоспособности(рис. **¿fig:022?**)



Просмотр регистров, для поиска ошибки в программе из листинга 10.3 (рис. **¿fig:023?**) и (рис. **¿fig:024?**)

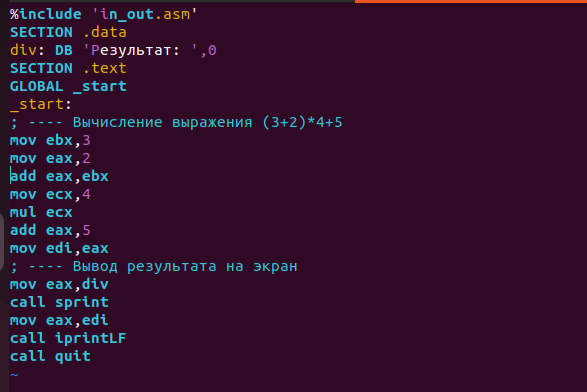




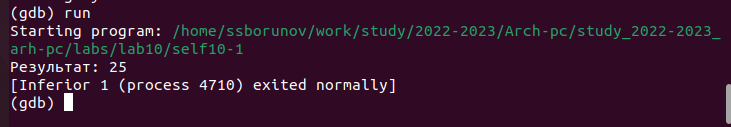
Ошибка была в сторках

add ebx,eax  
mov ecx,4  
mul ecx  
add ebx,5  
mov edi,ebx

правильно работающая программа представлена на (рис. **¿fig:025?**)



Проверка корректронсти работы программы, после исправлений (рис. **¿fig:026?**)



# 4 Выводы

В результате выполнения работы, я научился организовывать код в подпрограммы и познакомился с базовыми функциями отладчика gdb.